

IEEE Standard for Information Technology—Directory Services C Language Interfaces—Binding for Application Program Interface (API)

Sponsor

**Portable Applications Standards Committee
of the
IEEE Computer Society**

Approved March 18, 1993

IEEE Standards Board

Approved September 16, 1993

American National Standards Institute

Abstract: A C programming language binding for the Application Program Interface (API) to directory services in IEEE Std 1224.2-1993 is defined. This standard is intended for use by application developers, system implementors, test method writers, and users to support application portability at the source-code level. The interface is designed for operational, rather than for management, interactions with a directory such as knowledge management or schema management. Security features are not generally visible in the interface in order to permit flexibility in security policies.

Keywords: API, application portability, C language, directory, portable application, open systems

The Institute of Electrical and Electronics Engineers, Inc.

345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1994 by the

Institute of Electrical and Electronics Engineers, Inc.

All rights reserved. Published 1994

Printed in the United States of America

ISBN 1-55937-313-X

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretation is brought to the attention of the IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, the IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331

IEEE Standards documents are adopted by the Institute of
Electrical and Electronics Engineers without regard
to whether their adoption may involve patents
on articles, materials, or processes.
Such adoption does not assume any liability to any patent owner,
nor does it assume any obligation whatever to parties adopting
the standards documents.

Contents

	PAGE
Section 1: General	1
1.1 Scope	1
1.2 Normative References	1
1.3 Conformance	2
1.4 Test Methods	3
Section 2: Terminology and General Requirements	5
2.1 Conventions	5
2.2 Definitions	11
Section 3: The Generic Interface	23
3.1 Introduction	23
3.2 Interface Functions	24
3.2.1 Overview	24
3.2.2 Common Datatypes	24
3.2.3 Abandon	24
3.2.4 Add Entry	25
3.2.5 Bind	25
3.2.6 Compare	25
3.2.7 Initialize	25
3.2.8 List	26
3.2.9 Modify Entry	26
3.2.10 Modify RDN	26
3.2.11 Read	27
3.2.12 Receive Result	27
3.2.13 Remove Entry	27
3.2.14 Search	27
3.2.15 Shutdown	28
3.2.16 Unbind	28
3.2.17 Version	28
3.3 Directory Service Package	29
3.3.1 Object Identifiers	29
3.3.2 Enumeration Types and Constants	30
3.3.3 OM Object Constants	32
3.3.4 Integer Constants	33
3.3.5 Class Constraints	34
3.4 The <xds.h> Header	34
Section 4: The X.500 Directory Service	35
4.1 Overview	35
4.2 Basic Directory Contents Package	35

4.2.1	Directory Object and Attribute Object Identifiers	35
4.2.2	OM Object and Attribute Type Identifiers	36
4.2.3	Directory Enumeration Types and Constants	38
4.2.4	Integer Constants	38
4.3	Strong Authentication Package	38
4.3.1	Directory Object and Attribute Object Identifiers	38
4.3.2	OM Object and Attribute Type Identifiers	40
4.3.3	Enumeration Types and Constants	41
4.3.4	Integer Constants	41
4.4	MHS Directory User Package	41
4.4.1	Directory Object and Attribute Object Identifiers	41
4.4.2	OM Object and Attribute Type Identifiers	42
4.4.3	Enumeration Types and Constants	43
4.5	Header Files	43
4.5.1	The <xdsbdc ~~p~~ .h> Header	43
4.5.2	The <xdssap.h> Header	44
4.5.3	The <xdsmdup.h> Header	44
Section 5: Definitions of Constants		45
5.1	Introduction	45
5.2	Directory Service Package	45
5.2.1	Object and Attribute Type Identifiers	45
5.2.2	Enumeration Types and Constants	45
5.2.3	OM Object Constants	48
5.2.4	Integer Constants	48
5.3	Basic Directory Contents Package	49
5.4	Strong Authentication Package	49
5.5	MHS Directory User Package	50
Annex A (informative) Example Header Files		51
A.1	Introduction	51
A.2	Example <xds.h> Header	51
A.3	Example <xdsbdc ~~p~~ .h> Header	58
A.4	Example <xdssap.h> Header	61
A.5	Example <xdsmdup.h> Header	62
Annex B (informative) Example Programs		65
B.1	Introduction	65
B.2	Synchronous Directory Example	65
B.3	Asynchronous Directory Example	69
B.4	Example Existing Practice Directory Service Implementation	72
B.5	Error Handling Module	80
Annex C (informative) Bibliography		83
Alphabetic Topical Index		85

TABLES

Table 2-1 – C Naming Conventions	8
Table 4-1 – BDCP Directory Attribute Types	36
Table 4-2 – BDCP Directory Object Classes	37
Table 4-3 – Preferred Delivery Method Values	38
Table 4-4 – BDCP Upper Bounds	39
Table 4-5 – SAP Directory Attribute Types	39
Table 4-6 – SAP Directory Object Classes	40
Table 4-7 – SAP Upper Bounds	41
Table 4-8 – MDUP Directory Attribute Types	42
Table 4-9 – MDUP Directory Object Classes	42
Table 5-1 – OM Object Constant Names	48
Table 5-2 – Search Subset Constant Names	48
Table 5-3 – Receive Result Flag Constant Names	49
Table B-1 – OM Attributes of a Hostent	74

Introduction

(This introduction is not a normative part of IEEE Std 1327.2-1993, IEEE Standard for Information Technology—Directory Services C Language Interfaces—Binding for Application Program Interface (API), but is included for information only.)

Purpose

The purpose of this standard is to define a C programming language binding for the Application Program Interface (API) to Directory Services that is defined in IEEE Std 1224.2-1993 {4}.

A directory is a distributed collection of information, which programs can access in order to make queries or updates. IEEE Std 1224.2-1993 {4} defines, in programming language independent terms, an API to directory services. This API is known as the Directory Services API (DS API).

The DS API is intended to be used to provide access to a range of directory services that are instances of a common abstract model. That model is defined in the 1988 CCITT X.500 Series recommendations and ISO/IEC 9594: 1990. IEEE Std 1224.2-1993 {4} prescribes how the DS API is to be used to access the particular directory service defined in ISO/IEC 9594: 1990 and indicates how it may be used to access other directory services that conform to the same abstract model. Nothing in IEEE Std 1224.2-1993 {4} or in this standard requires that the implementation of the interface or the Directory itself actually make use of the Directory Access Protocol (DAP), the Directory System Protocol (DSP), or other parts of the model, just so long as it provides the defined service.

The interface is designed for operational interactions with a directory, rather than for management interactions such as knowledge management or schema management. Also, security features are not generally visible in the interface in order to permit flexibility in security policies. It is intended that an application program should be able to use the interface to access a single directory service or to access several directory services at the same time.

Related Standards

The programming language independent specification on which the programming language binding defined in this standard is based is contained in IEEE Std 1224.2-1993 {4}.

This standard is intended to be used in conjunction with the C programming language binding defined in IEEE Std 1327-1993 {6} for the OSI abstract data Manipulation (OM) API defined in IEEE Std 1224-1993 {3}.

IEEE Std 1328.2-1993 {7} specifies a set of requirements to be satisfied by test methods for measuring conformance to this standard. In addition, such test methods must also satisfy the requirements specified in IEEE Std 1326.2-1993 {5} for

1 test methods for measuring conformance to the programming language indepen-
2 dent specification of the API.

3 ISO/IEC 9954: 1990 and the CCITT X.500 Series recommendations define the gen-
4 eral model for the directory services to which the API defined in this standard pro-
5 vides access.

6 The API is intended to be used, if required, in conjunction with the C language
7 binding defined in IEEE Std 1327.1-1993 {B15} to the API to X.400-based elect-
8 ronic messaging defined in IEEE Std 1224.1-1993 {B14}.

9 **Overview**

10 The API defined in this standard contains three major parts. The first part is
11 mandatory; it consists of the interface for accessing the services defined in the pro-
12 gramming language independent specification, together with definitions for the
13 information types required to access any directory service. The second part is
14 optional; it consists of definitions for the information types required to access the
15 particular directory service defined in X.500 directory standards, and is defined in
16 Section 4. The third part is also optional; it contains definitions of symbolic con-
17 stants that are required to ensure interworking with implementations of related
18 services, such as the API to X.400-based electronic messaging defined in IEEE Std
19 1327.1-1993 {B15}.

20 **Base Document**

21 The base document for this standard is the X/Open Directory Services API
22 Specification (XDS) {B17}, copyright © 1991 by X.400 API Association, United
23 States of America and X/Open Company Limited, United Kingdom. The working
24 group is grateful to both X/Open and the X.400 API Association for permission to
25 use their materials.

26 This standard was prepared by the Namespace and Directory Services Working
27 Group (P1224.2, formerly P1003.17), sponsored by the Portable Applications Stan-
28 dards Committee of the IEEE Computer Society. At the time this standard was
29 approved, the membership of the P1224.2 working group was as follows:

1
2
3
4
5
6
7
8

9

10
11
12
13

14

15
16

17
18

19
20

21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39

Portable Applications Standards Committe (PASC)

Chair: Jim Isaak
Vice Chairs: Hal Jespersen
Lorraine Kevra
Pete Meier
Andrew Twigger
Treasurer: Peter Smith
Secretary: Shane McCarron

P1224.2 Working Group Officials

Chair: Mark Hazzard
Vice Chair: Donna Fisher
Editor: Chris Harding
Secretary: Donna Fisher

P1224.2 Working Group

Greg Bussiere	Edmun Liu	Rob Spade
Scott Guthery	Ted Roberts	Alec Tashjian

The following persons were members of the balloting group that approved this standard for submission to the IEEE Standards Board:

Stephen R. Walli	<i>EurOpen Organizational Representative</i>	
Ralph Barker	<i>Uniforum Organizational Representative</i>	
Reed M. Adams	Michel Gien	Richard Seibel
Charles R. Arnold	Steve Griesmer	Ronald K. Sellars
Timothy Baker	Charles Hammons	Rob Spade
Ralph Barker	Michael Hannah	Sandra Swearingen
Andy Bihain	C. J. Harding	James G. Tanner
Mack W. Bishop	Mark Hazzard	Alec Tashjian
Robert Bismuth	Barry Hedquist	Donn S. Terry
Kevin Brady	Khiem Ho	Alan E. Turner
Dawn Burnett	Petr Janeček	Andrew T. Twigger
Gregory Bussiere	Hal Jespersen	Mark-Rene Uchida
George Carson	Robert Knighten	Andrew Walker
Andy Cheese	Kevin Lewis	Paul Wanish
Fred D. Crouner	Roger Martin	Peter Wishart
Ana Maria De Alvare	Mark Modig	Oren Yuen
Stephen L. Diamond	Diana Norwood	John Zenor
Philip H. Enslow	Edward Owens	George R. Zerdian
Donna K. Fisher	Scott E. Preece	Jason Zions
Donald Folland	Paul Rabin	John Zolnowsky
	Seth Rosenthal	

1 The IEEE Standards Board approved this standard on March 18, 1993 with the
2 following membership:

3 **Wallace S. Read, *Chair***

Donald C. Loughry, *Vice Chair*

4 **Andrew G. Salem, *Secretary***

5 Gilles A. Baril

Ben C. Johnson

Don T. Michael*

6 Clyde R. Camp

Walter J. Karplus

Marco W. Migliaro

7 Donald C. Fleckenstein

Lorraine C. Kevra

L. John Rankine

8 Jay Forster*

E. G. "Al" Kiener

Arthur K. Reilly

9 David F. Franklin

Ivor N. Knight

Ronald H. Reimer

10 Ramiro Garcia

Joseph Koepfinger*

Gary S. Robinson

11 Donald N. Heirman

D. N. "Jim" Logothetis

Leonard L. Tripp

12 Jim Isaak

Donald W. Zipse

13 *Member Emeritus

14 Also included are the following nonvoting IEEE Standards Board liaisons:

15 Satish K. Aggarwal

16 James Beall

17 Richard B. Engelman

18 David E. Soffrin

19 Stanley I. Warshaw

20 Mary Lynne Nielsen

21 *IEEE Standards Project Editor*

IEEE Standard for Information Technology—Directory Services C Language Interfaces—Binding for Application Program Interface (API)

Section 1: General

1.1 Scope

This standard defines an ISO/IEC 9899: 1990 {1} C language binding to the language-independent API defined in IEEE Std 1224.2-1993 {4}.¹⁾ It is intended to support application portability at the source-code level and to be used by application developers, system implementors, test method writers, and users.

1.2 Normative References

The following standards contain provisions that, through references in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

{1} ISO/IEC 9899:1990,²⁾ *Programming Languages—C*.

1) The numbers in curly brackets correspond to those of the references in 1.2. When preceded by a “B,” the numbers correspond to those of the bibliography in Annex C.

2) ISO/IEC documents can be obtained from the ISO office, 1 Rue de Varembe, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse.

- 1 {2} ISO/IEC 9945-1: 1990 (IEEE Std 1003.1: 1990),³⁾ *Information technology—*
2 *Portable Operating System Interface (POSIX)—Part 1: System Application*
3 *Program Interface (API) [C Language].*
- 4 {3} IEEE Std 1224-1993, *IEEE Standard for Information Technology—Open*
5 *Systems Interconnection (OSI) Abstract Data Manipulation—Application*
6 *Program Interface (API) [Language Independent].*
- 7 {4} IEEE Std 1224.2-1993, *IEEE Standard for Information Technology—*
8 *Directory Services—Application Program Interface (API) [Language Inde-*
9 *pendent].*
- 10 {5} IEEE Std 1326.2-1993, *IEEE Standard for Information Technology—Test*
11 *Methods for Measuring Conformance to Directory Services—Application Pro-*
12 *gram Interface (API) [Language Independent].*
- 13 {6} IEEE Std 1327-1993, *IEEE Standard for Information Technology—Open*
14 *Systems Interconnection (OSI) Abstract Data Manipulation C Language*
15 *Interfaces—Binding for Application Program Interface (API).*
- 16 {7} IEEE Std 1328.2-1993, *IEEE Standard for Information Technology—Test*
17 *Methods for Measuring Conformance to Directory Services C Language*
18 *Interfaces—Binding for Application Program Interface (API).*

19 1.3 Conformance

20 1.3.1 Implementation Conformance

21 1.3.1.1 Requirements

22 A conforming implementation for this standard shall be a conforming implementa-
23 tion, as defined in IEEE Std 1224.2-1993 {4}, for the programming language bind-
24 ing specification that is contained in this standard.

25 1.3.1.2 Documentation

26 A conformance document shall be available for an implementation claiming confor-
27 mance to this standard. It shall satisfy the requirements that are defined in IEEE
28 Std 1224.2-1993 {4} for a conformance document for a programming language
29 binding specification.

30
31 3) IEEE publications are available from the Institute of Electrical and Electronics Engineers, Service Center, 445
32 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

The phrases “shall document” or “shall be documented” in this standard mean that documentation of the feature shall appear in the conformance document, unless the system documentation is explicitly mentioned.

The system documentation should also contain the information found in the conformance document.

1.3.1.3 Conforming Implementation Options

The use of the symbolic constant values defined in Section 5 is optional. If an implementation claims to work with other implementations, then it shall use the symbolic constant values defined in Section 5 and shall also use the optional symbolic constant values defined in IEEE Std 1224.2-1993 {4}.

1.3.2 Application Conformance

All applications claiming conformance to this standard shall fall within one of the categories in the following subclauses.

1.3.2.1 Strictly Conforming Application

A Strictly Conforming Application for this standard is a Strictly Conforming Application as defined in IEEE Std 1224.2-1993 {4} for the programming language binding specification contained in this standard.

1.3.2.2 Conforming Application

There is only one type of Conforming Application.

1.3.2.2.1 ISO/IEC Conforming Application

An ISO/IEC Conforming Application for this standard is an ISO/IEC Conforming Application as defined in IEEE Std 1224.2-1993 {4} for the programming language binding specification contained in this standard.

1.3.2.3 Conforming Application Using Extensions

A Conforming Application Using Extensions for this standard is a Conforming Application Using Extensions as defined in IEEE Std 1224.2-1993 {4} for the programming language binding specification contained in this standard.

1.4 Test Methods

Any measurement of conformance to this standard shall be performed using test methods that conform to IEEE Std 1326.2-1993 {5} and IEEE Std 1328.2-1993 {7}.

Section 2: Terminology and General Requirements

2.1 Conventions

2.1.1 General and Typographical Conventions

Many technical terms, such as “object” and “attribute,” are used by both OSI abstract data manipulation and directory services descriptions. In the first case, they refer to constructs of the OSI abstract data manipulation interface, while in the second case they refer to constructs of the directory service to which the interface provides access. The meanings ascribed to these terms in these cases are often similar but different. Throughout this standard, care is taken to distinguish between these meanings, for example by qualifying the term with “OM” or “directory” as appropriate (as with OM classes and directory classes and with OM attributes and directory attributes). The unqualified term “attribute” denotes the directory construct, while the phrase “OM attribute” denotes the OSI abstract data manipulation construct. The phrase “Object Class” denotes the directory construct, while the phrase “OM class” denotes the OSI abstract data manipulation construct.

The term “operation” is used in two different senses, one by the language-independent specification conventions and one by the directory service descriptions. When used in the sense of the directory, it is always qualified by the term “directory,” in the phrase “directory operation.” The unqualified term “operation” denotes the language-independent specification construct. Sometimes, to reinforce the distinction, the phrase “interface operation” is used to denote the language-independent specification construct, instead of “operation.”

The reader is urged to be cautious, and reference to 2.2 (Definitions) may be useful.

Language-independent concrete OM class names, OM attribute names, and OM attribute value names appear in bold font while abstract OM class names appear in bold italic font. They are all spelled with hyphens between words. The first letters of language-independent OM class and OM attribute names are capitalized (e.g., **Filter-Item**).

Language-independent datatype, operation argument, and error names appear in Helvetica font, are lowercased, and are spelled with underscores between words (e.g., `ds_abandon`).

Each OM class has an associated datatype whose name is obtained from the class name by converting uppercase to lowercase, converting hyphens to underscores, adding the prefix “ds_”, and adding the suffix “_type”. (Thus, `ds_filter_item_type` is the datatype corresponding to OM class **Filter-Item**.)

Function synopses and other extended pieces of C code appear in constant width (courier) font while C language names embedded in ordinary text appear in italic font. Underscores are used to separate words in C language names. The C language names are derived mechanically from the corresponding language-independent names in a manner described in 2.1.4.

A C language function, datatype, or code fragment appears in *italic* font. A function name is indicated by following parentheses [e.g., *ds_abandon()*].

A C language symbolic constant, other than an error name, is surrounded by braces (e.g., {DS_MAX_OUTSTANDING_OPERATIONS}).

A C language symbolic constant for an error name is surrounded by brackets (e.g., [DS_E_ADMIN_LIMIT_EXCEEDED]).

When important terms are introduced, they appear in italics. Italics are also used in 2.2 for cross-references.

The use of fonts in this standard is as follows:

(1) The Helvetica font is used for

- Language-independent operation names, such as `ds_add_entry`
- Language-independent datatype names, such as `ds_status_type`
- Language-independent error names, such as Abandon-Failed

(2) The *italic* font is used for

- Language-independent operation arguments, such as *session*
- C language names embedded in ordinary text
- The introduction of important terms
- Cross-references in 2.2

(3) The **bold** font is used for

- Language-independent concrete OM class names, such as **Filter-Item**
- Language-independent OM attribute names, such as **Filter-Item-Type**
- Language-independent OM attribute values, such as **approximate-match**

(4) The **bold italic** font is used for

- Language-independent abstract OM class names, such as ***Common-Results***

- 1 (5) The **constant width** (Courier) font is used for
2 — Terms defined in standards upon which this standard is based
3 — Function synopses and other extended pieces of C code

4 **2.1.2 Representation of Strings of Numbers**

5 Strings of numbers (such as ASN.1 Object Identifier encodings) are represented in
6 the form "`\xaa\xbb\xcc . .`", where each term `\xnn` represents an element of the
7 string, and the value of that element expressed in hexadecimal notation is `nn`. So,
8 for example, the string consisting of the three numbers 1, 10, 100 is represented as
9 "`\x01\x0a\x64`".

10 **2.1.3 Language-Independent Conventions**

11 The language-independent specification for this standard is contained in IEEE Std
12 1224.2-1993 {4}. The language-independent specification conventions apply to that
13 standard and to language specification terms used in this standard. These con-
14 ventions are described in IEEE Std 1224-1993 {3}.

15 **2.1.4 C Language Binding Conventions**

16 **2.1.4.1 Introduction**

17 This standard specifies C identifiers for all the elements of the interface, so that
18 application programs written in C can access the directory. These elements
19 include function names, typedef names, and constants. All the C identifiers are
20 mechanically derived from the language-independent names as explained below.

21 **2.1.4.2 C Naming Conventions**

22 The interface uses part of the C public namespace for its facilities. All identifiers
23 start with the letters *ds*, *DS*, or *OMP*. More detail about the conventions used is
24 given in Table 2-1. The interface reserves all identifiers starting with the letters
25 *dsP* for private (i.e., internal) use by implementations of the interface. It also
26 reserves *all* identifiers starting with the letters *dsX* or *DSX* for extensions of the
27 interface. If the `<xds.h>` header is included, the application shall not use any
28 identifier starting with these letters.

29 IEEE Std 1224-1993 {3} uses similar, though not identical, naming conventions.
30 All OSI abstract data manipulation identifiers are prefixed by the letters *OM* or
31 *om*.

32 No implementation of the interface shall use any public identifiers other than
33 those defined in this standard and those beginning *dsP*, *dsX*, *DSX*, or *OMP*. A
34 *public identifier* is any name except those reserved in the C Standard (ISO/IEC

Table 2-1 – C Naming Conventions

Item	Prefix
Reserved for implementors	<i>dsP</i>
Reserved for interface extensions	<i>dsX</i>
Reserved for interface extensions	<i>DSX</i>
Reserved for implementors	<i>OMP</i>
Functions	<i>ds_</i>
Error “problem” values	<i>DS_E_</i>
OM class names	<i>DS_C_</i>
OM value length limits	<i>DS_VL_</i>
OM value number limits	<i>DS_VN_</i>
Other constants	<i>DS_</i>
Attribute Type	<i>DS_A_</i>
Object Class	<i>DS_O_</i>

9899: 1990 {1}), and the *public namespace* is the set of all possible public identifiers.

The C identifiers are derived from the language-independent names used throughout this standard by a purely mechanical process that depends on the kind of name.

— The C function names are identical to the language-independent operation names. Within text, they are italicized and followed by “()”.

Thus

— *ds_receive_result*

becomes

— *ds_receive_result()*

— C function input parameters are identical to the corresponding language-independent argument names. C function output parameters are derived from the argument names by adding “_return” as a suffix.

Thus, the output argument

— *operation_status*

becomes

— *operation_status_return*

(Within text, both language-independent argument names and C function parameter names are italicized.)

— The names of constants identifying OM classes are derived from the class names by converting lowercase to uppercase, converting hyphens to underscores, and adding the prefix “DS_C_”.

- 1 Thus
- 2 — **Read-Result**
- 3 becomes
- 4 — {DS_C_READ_RESULT}
- 5 — The names of C language constants that denote language-independent
6 specification choice values are derived from the choice value names by con-
7 verting lowercase to uppercase and adding the prefix “DS_”.
8 Thus
- 9 — **Default-Context**
- 10 becomes
- 11 — {DS_DEFAULT_CONTEXT}
- 12 — Enumeration tags are derived from the name of the corresponding OM syn-
13 tax by adding the prefix “DS_”. Hyphens are converted to underscores, but
14 the case of letters is left unchanged.
15 Thus
- 16 — **Enum(Limit-Problem)**
- 17 becomes
- 18 — *DS_Limit_Problem*
- 19 — For enumeration constants, OM attributes, and all other constants except
20 errors, lowercase is converted to uppercase, hyphens are converted to
21 underscores, and the prefix “DS_” is added.
22 Thus
- 23 — **O-Residential-Person**
- 24 becomes
- 25 — {DS_O_RESIDENTIAL_PERSON}
- 26 — Errors are treated as a special case. Constants that are the possible values
27 of the OM attribute **Problem** of a subclass of the OM class **Error** have
28 hyphens converted to underscores, are made entirely uppercase, and are
29 prefixed by “DS_E_”.
30 Thus
- 31 — *alias-dereferencing-problem*
- 32 becomes
- 33 — [DS_E_ALIAS_DEREFERENCING_PROBLEM]
- 34 — The constants in the “Value Length” and “Value Number” columns of the
35 OM class definition tables are also assigned identifiers. (They have no
36 names in IEEE Std 1224.2-1993 {4}.) When the upper limit in one of these

1 columns is not “1” (one), it is given a name consisting of the OM attribute
2 name prefixed by “DS_VL_” for value length or “DS_VN_” for value
3 numbers.

4 — The sequence of octets for each object identifier is also assigned an identifier
5 for internal use by certain OM macros. These identifiers are all uppercase
6 and are prefixed by “OMP_O_”. See IEEE Std 1224-1993 {3} for further
7 details on the use of object identifiers.

8 Note that hyphens are translated to underscores.

9 2.1.4.3 Use and Implementation of Interfaces

10 If an argument to a function has an invalid value (such as a value outside the
11 domain of the function, a pointer outside the address space of the program, or a
12 null pointer), the behavior is undefined.

13 Any function declared in a header may be implemented as a macro defined in the
14 header, so a library function should not be declared explicitly if its header is
15 included. Any macro definition of a function can be suppressed locally by enclosing
16 the name of the function in parentheses, because the name is not then followed
17 by the left parenthesis that indicates expansion of a macro function name.
18 For the same syntactic reason, it is permitted to take the address of a library function
19 even if it is also defined as a macro. The use of *#undef* to remove any macro
20 definition will also ensure that the identifier refers to an actual function. Any
21 invocation of a library function that is implemented as a macro shall expand to
22 code that evaluates each of its arguments exactly once, fully protected by parentheses
23 where necessary, so it is generally safe to use arbitrary expressions as
24 arguments. Likewise, any function-like macros can be invoked in an expression
25 anywhere a function with a compatible return type could be called.

26 These concepts apply in this standard unless explicitly stated otherwise in the C
27 language binding descriptions.

28 2.1.4.4 Function Return Values

29 Except for *ds_initialize()*, the return value of a C function is bound to the *Status*
30 result of the language-independent description. Functions shall return a value of
31 type *DS_status*, which is an error indication. If and only if the function succeeds,
32 its value shall be **success**, expressed in the C language by the constant
33 {DS_SUCCESS}. If a function returns a status other than this, then it shall not
34 have updated the return parameters. The value of the status, in this case, shall
35 be an error as described in IEEE Std 1224.2-1993 {4}.

36 Since the C language does not provide multiple return values, functions shall
37 return all other results by writing them into storage passed by the application
38 program. Any argument that is a pointer to such storage has a name ending with
39 *_return*. For example, the C parameter declaration

40 — *om_uint * completion_flag_return*

indicates that the function shall return an unsigned integer *Completion-Flag* as a result, so the actual argument to the function shall be the address of a suitable variable. This notation allows the reader to distinguish between an input parameter that happens to be a pointer and an output parameter where the “*” is used to simulate the semantics of passing by reference.

2.2 Definitions

2.2.1 Terminology

For the purposes of this standard, the following definitions apply.

2.2.1.1 conformance document: A document provided by an implementor that contains implementation details as described in 1.3.1.2. [ISO/IEC 9945-1: 1990 {2}]

2.2.1.2 implementation defined: An indication that the implementation shall define and document the requirements for correct program constructs and correct data of a value or behavior. [ISO/IEC 9945-1: 1990 {2}]

2.2.1.3 locally administered: An indication that the configuration is not consistent across all systems and that the administrator of each system shall provide documentation of its behavior. [IEEE Std 1224.2-1993 {4}]

2.2.1.4 may: An indication of an optional feature.

With respect to implementations, the word *may* is to be interpreted as an optional feature that is not required in this standard, but that can be provided. With respect to Strictly Conforming Applications, the word *may* means that the optional feature shall not be used. [ISO/IEC 9945-1: 1990 {2}]

2.2.1.5 shall: An indication of a requirement on the implementation or on Strictly Conforming Applications, where appropriate. [ISO/IEC 9945-1: 1990 {2}]

2.2.1.6 should:

- (1) With respect to implementations, an indication of an implementation recommendation, but not a requirement.
- (2) With respect to applications, an indication of a recommended programming practice for applications and a requirement for Strictly Conforming Applications. [ISO/IEC 9945-1: 1990 {2}]

2.2.1.7 supported: A condition regarding optional functionality.

Certain functionality in this standard is optional, but the interfaces to that functionality are always required. If the functionality is *supported*, the interfaces work as specified in this standard (except that they do not return the error condition indicated for the unsupported case). If the functionality is not *supported*, the interface shall always return the indication specified for this situation. [ISO/IEC 9945-1: 1990 {2}]

2.2.1.8 system documentation: All documentation provided with an implementation, except the conformance document.

Electronically distributed documents for an implementation are considered part of the system documentation. [ISO/IEC 9945-1: 1990 {2}]

2.2.1.9 undefined: An indication that this standard imposes no portability requirements on an application's use of an indeterminate value or its behavior with erroneous program constructs or erroneous data.

Implementations (or other standards) may specify the result of using that value or causing that behavior. An application using such behaviors is using extensions, as defined in 1.3.2.3. [ISO/IEC 9945-1: 1990 {2}]

2.2.1.10 unspecified: An indication that this standard imposes no portability requirements on applications for correct program constructs or correct data regarding a value or behavior.

Implementations (or other standards) may specify the result of using that value or causing that behavior. An application requiring a specific behavior, rather than tolerating any behavior when using that functionality, is using extensions, as defined in 1.3.2.3. [ISO/IEC 9945-1: 1990 {2}]

2.2.2 General Terms

For the purposes of this standard, the following definitions apply.

2.2.2.1 abstract class: An OM class of OM objects of which instances are forbidden. [IEEE Std 1224.2-1993 {4}]

2.2.2.2 Abstract Syntax Notation One (ASN.1): A notation that both enables complicated types to be defined and also enables values of these types to be specified (see ISO/IEC 8824: 1990 {B1}). [IEEE Std 1224.2-1993 {4}]

2.2.2.3 access point: The point at which an abstract service is obtained. [ISO/IEC 9594-2: 1990 {B7}]

2.2.2.4 address: An unambiguous name, label, or number that identifies the location of a particular entity or service. [IEEE Std 1224.2-1993 {4}]

See also *presentation address* (2.2.2.66).

2.2.2.5 alias [alias name]: An alternate name for a directory object, provided by the use of one or more alias entries in the DIT. [ISO/IEC 9594-2: 1990 {B7}]

2.2.2.6 alias entry: A Directory entry, of Object Class “alias,” containing information used to provide an alternative name for an object. [IEEE Std 1224.2-1993 {4}]

2.2.2.7 alias name: See *alias* in 2.2.2.5.

2.2.2.8 argument: Information that is passed to an interface operation or a directory operation. [IEEE Std 1224.2-1993 {4}]

2.2.2.9 asynchronous completion: Completion of an asynchronous directory operation.

An asynchronous directory operation is complete when a corresponding synchronous directory operation would complete and any associated status fields have been updated. [IEEE Std 1224.2-1993 {4}]

2.2.2.10 asynchronous operation: An operation that does not of itself cause the process requesting the operation to be blocked from further use of the CPU.

This implies that the process and the operation are running concurrently. [IEEE Std 1224.2-1993 {4}]

2.2.2.11 attribute: Used alone, means *directory attribute* as in 2.2.2.28 (see also *OM attribute*, 2.2.2.57).

2.2.2.12 attribute syntax [directory syntax]: A definition of the set of values that a directory attribute may assume.

It includes the datatype, in ASN.1, and, usually, one or more matching rules by which values may be compared. [IEEE Std 1224.2-1993 {4}]

2.2.2.13 attribute type: Used alone, means *directory attribute type* as in 2.2.2.29 (see also *OM attribute type*, 2.2.2.58).

2.2.2.14 attribute value: Used alone, means *directory attribute value* as in 2.2.2.30 (see also *OM attribute value*, 2.2.2.59).

1 **2.2.2.15 attribute value assertion:** A proposition, which may be true, false, or
2 undefined, about the values (or perhaps only the distinguished values) of an entry.
3 [ISO/IEC 9594-2: 1990 {B7}]

4 NOTE: An attribute value assertion consists of an attribute type and a single value. Loosely, it is
5 true if one of the values of the given attribute in the entry matches the given value.

6 **2.2.2.16 attribute value syntax:** See *OM syntax* in 2.2.2.62.

7 **2.2.2.17 basic encoding rules:** A specific set of rules used to encode ASN.1 val-
8 ues as strings of octets. [IEEE Std 1224.2-1993 {4}]

9 **2.2.2.18 cache:** See *copy* in 2.2.2.23.

10 **2.2.2.19 certificate:** A synonym for *user certificate*, 2.2.2.84.

11 **2.2.2.20 chaining:** A mode of interaction optionally used by a DSA that cannot
12 perform a directory operation itself; the DSA chains by invoking a directory opera-
13 tion of another DSA and then relaying the outcome to the original requester.
14 [ISO/IEC 9594-4: 1990 {B9}]

15 **2.2.2.21 class:** See *object class* in 2.2.2.54 and *OM class* in 2.2.2.60.

16 **2.2.2.22 continuation reference:** A reference that describes how the perfor-
17 mance of all or part of a directory operation can be continued at a different DSA or
18 DSAs. [IEEE Std 1224.2-1993 {4}]

19 See also *referral* (2.2.2.71).

20 **2.2.2.23 copy:**

21 (1) A copy of an entry stored in other DSA(s) through bilateral
22 agreement,

23 (2) A locally and dynamically stored copy of an entry resulting from a
24 request (a cache copy). [IEEE Std 1224.2-1993 {4}]

25 **2.2.2.24 concrete class:** An OM class of which instances are permitted. [IEEE
26 Std 1224.2-1993 {4}]

27 **2.2.2.25 descriptor:** The means by which the client and service exchange an OM
28 attribute value, and the integers that denote its representation, type, and syntax.
29 [IEEE Std 1224.2-1993 {4}]

30 **2.2.2.26 descriptor list:** An ordered sequence of descriptors that is used to rep-
31 resent several OM attribute types and values. [IEEE Std 1224.2-1993 {4}]

1 **2.2.2.27 directory:** A repository of information about objects that provides direc-
2 tory services to its users, which allows its users access to the information.

3 **2.2.2.28 directory attribute [attribute]:** The information of a particular type
4 concerning an object and appearing in an entry describing the object in the DIB.
5 [ISO/IEC 9594-2: 1990 {B7}]

6 **2.2.2.29 directory attribute type [attribute type]:** That component of a *direc-*
7 *tory attribute* that indicates the class of information given by that attribute.

8 It is an *object identifier*, and so completely unique. [ISO/IEC 9594-2: 1990 {B7}]

9 **2.2.2.30 directory attribute value [attribute value]:** A particular instance of
10 the class of information indicated by a directory attribute type. [ISO/IEC 9594-2:
11 1990 {B7}]

12 **2.2.2.31 directory class:** See *object class* in 2.2.2.54.

13 **2.2.2.32 directory information base:** The complete set of information to which
14 the directory provides access and that includes all of the pieces of information that
15 can be read or manipulated using the operations of the directory. [ISO/IEC 9594-
16 2: 1990 {B7}]

17 **2.2.2.33 directory information tree:** The directory information base, consid-
18 ered as a tree, whose vertices (other than the root) are the directory entries.
19 [ISO/IEC 9594-2: 1990 {B7}]

20 **2.2.2.34 directory object:** Anything in some “world,” generally the world of
21 telecommunications and information processing or some part thereof, that is
22 identifiable (can be named) and that it is of interest to hold information on in the
23 directory information base. [ISO/IEC 9594-2: 1990 {B7}]

24 **2.2.2.35 directory operation:** Processing performed within the directory to pro-
25 vide a service, such as a read directory operation; it is given some arguments as
26 input, performs some processing, and returns some results.

27 An application process causes a directory operation to be performed by invoking an
28 interface operation. [IEEE Std 1224.2-1993 {4}]

29 **2.2.2.36 directory syntax:** See *attribute syntax* in 2.2.2.12.

30 **2.2.2.37 directory system agent:** An OSI application process that is part of the
31 Directory. [ISO/IEC 9594-2: 1990 {B7}]

2.2.2.38 directory user agent: An OSI application process that represents a user accessing the Directory. [ISO/IEC 9594-2: 1990 {B7}]

NOTE: This may be composed of an arbitrary number of system processes and application processes, including the one or more that are the user(s).

2.2.2.39 distinguished encoding: Restrictions to the basic encoding rules designed to ensure a unique encoding of each ASN.1 value, defined in clause 8.7 of ISO/IEC 9594-8: 1990 {B12}. [IEEE Std 1224.2-1993 {4}]

2.2.2.40 distinguished name: One of the names of an object, formed from the sequence of RDNs of its object entry and each of its superior entries. [ISO/IEC 9594-2: 1990 {B7}]

2.2.2.41 distinguished value: An attribute value in an entry that has been designated to appear in the relative distinguished name of the entry. [IEEE Std 1224.2-1993 {4}]

2.2.2.42 entry [object entry]: A part of the DIB that contains information about an object.

Each entry is made up of attributes. [ISO/IEC 9594-2: 1990 {B7}]

2.2.2.43 filter: An assertion about the presence or value of certain attributes of an entry in order to limit the scope of a search. [ISO/IEC 9594-3: 1990 {B8}]

2.2.2.44 function: A programming language construct, modeled after the mathematical concept.

A function encapsulates some behavior. It is given zero or more arguments as input, performs some processing, and returns some results. Functions are also known as procedures, subprograms, or subroutines. (See also *operation*, 2.2.2.63.) [IEEE Std 1224.2-1993 {4}]

2.2.2.45 immediate subordinate: An entry in the DIT that is immediately below another entry in the tree.

The distinguished name of the immediate subordinate is formed by appending its relative distinguished name to the distinguished name of the other entry. [IEEE Std 1224.2-1993 {4}]

2.2.2.46 immediate superior: An entry in the DIT that is immediately above another entry in the tree.

The distinguished name of the immediate superior, followed by the relative distinguished name of the other entry, forms the distinguished name of the other entry.

1 Each entry (except the root) has exactly one immediate superior. [IEEE Std
2 1224.2-1993 {4}]

3 **2.2.2.47 interface operation:** See *operation* in 2.2.2.63.

4 **2.2.2.48 invoke ID:** An identifier used to distinguish one directory operation
5 from all other outstanding operations. [IEEE Std 1224.2-1993 {4}]

6 **2.2.2.49 knowledge reference:** Knowledge that associates, either directly or
7 indirectly, a DIT entry with the DSA in which it is located. [ISO/IEC 9594-4: 1990
8 {B9}]

9 **2.2.2.50 leaf entry:** A Directory entry that has no subordinates.
10 It can be an alias entry or an object entry. [IEEE Std 1224.2-1993 {4}]

11 **2.2.2.51 name:** A construct that singles out a particular (directory) object from all
12 other objects.

13 A name shall be unambiguous (that is, denote just one object); however, it need
14 not be unique (that is, be the only name that unambiguously denotes the object).
15 [ISO/IEC 9594-2: 1990 {B7}]

16 **2.2.2.52 nonspecific subordinate reference:** A knowledge reference that holds
17 information about a DSA that holds one or more unspecified subordinate entries.
18 [ISO/IEC 9594-4: 1990 {B9}]

19 **2.2.2.53 object:** See *directory object* in 2.2.2.34 and *OM object* in 2.2.2.61.

20 **2.2.2.54 object class [directory class]:** An identified family of objects (or con-
21 ceivable objects) that share certain characteristics. [IEEE Std 1224.2-1993 {4}]

22 **2.2.2.55 object entry:** See *entry* in 2.2.2.42.

23 **2.2.2.56 object identifier:** A value (distinguishable from all other such values)
24 that is associated with an information object (see ISO/IEC 8824: 1990 {B1}).
25 [IEEE Std 1224.2-1993 {4}]

26 **2.2.2.57 OM attribute:** A component of an OM object, comprised of an integer
27 denoting the type of the attribute and an ordered sequence of one or more attri-
28 bute values, each accompanied by an integer denoting the syntax of the value.

29 **2.2.2.58 OM attribute type:** A category into which OM attribute values are
30 placed on the basis of their purpose.

1 **2.2.2.59 OM attribute value:** An arbitrarily complex information item that can
2 be viewed as a characteristic or property of an OM object.

3 **2.2.2.60 OM class:** A category into which OM objects are placed on the basis of
4 both their purpose and their internal structure.

5 **2.2.2.61 OM object:** Any of the complex information objects created, examined,
6 modified, or destroyed by means of the interface.

7 **2.2.2.62 OM syntax [attribute value syntax]:** A category into which an OM
8 attribute value is placed on the basis of its form.

9 **2.2.2.63 operation:** A program abstraction with formal input and output param-
10 eters that are bound to objects or values in the application that invokes it. [IEEE
11 Std 1224-1993 {3}]

12 **2.2.2.64 outstanding directory operation:** A directory operation, invoked
13 asynchronously (i.e., with **asynchronous** = **true** in the context), that has not yet
14 been the subject of an invocation of the `ds_abandon` interface operation or the
15 `ds_receive_result` interface operation. [IEEE Std 1224.2-1993 {4}]

16 **2.2.2.65 package:** A group of related OM classes.

17 **2.2.2.66 presentation address:** An unambiguous name that is used to identify
18 a set of presentation service access points.

19 Loosely, it is the network address of an OSI service. [IEEE Std 1224.2-1993 {4}]

20 **2.2.2.67 private OM object:** The internal representation of the language-
21 independent model of an OM object in a service, and thus unspecified. [IEEE Std
22 1224-1993 {3}]

23 **2.2.2.68 process:** An address space, a single thread of control that executes
24 within that address space, and its required system resources.

25 On a system that implements *threads*, a process is redefined to consist of an
26 address space with one or more threads executing within that address space and
27 their required system resources. [ISO/IEC 9945-1: 1990 {2}]

28 NOTE: The term *process* is used in contrast to “system process,” or the OSI usage of the term “appli-
29 cation process.”

30 **2.2.2.69 public OM object:** The representation by the client of an object in a
31 particular programming language binding. [IEEE Std 1224-1993 {3}]

- 1 **2.2.2.70 purported name:** A construct that is syntactically a name but that has
2 not (yet) been shown to be a valid name. [ISO/IEC 9594-2: 1990 {B7}]
- 3 **2.2.2.71 referral:** An outcome that can be returned by a DSA that cannot per-
4 form a directory operation itself, and that identifies one or more other DSAs more
5 able to perform the directory operation. [ISO/IEC 9594-4: 1990 {B9}]
- 6 **2.2.2.72 relative distinguished name:** A set of AVAs, each of which is true,
7 concerning the distinguished values of a particular entry. [ISO/IEC 9594-2: 1990
8 {B7}]
- 9 **2.2.2.73 replication:** The process by which copies of entries are created and
10 maintained. [IEEE Std 1224.2-1993 {4}]
- 11 **2.2.2.74 result:** Information that is returned from an interface operation or a
12 directory operation and that constitutes the outcome of the processing that was
13 performed. [IEEE Std 1224.2-1993 {4}]
- 14 **2.2.2.75 schema:** The set of rules and constraints concerning DIT structure,
15 object class definitions, attribute types, and syntaxes that characterize the DIB.
16 [ISO/IEC 9594-2: 1990 {B7}]
- 17 **2.2.2.76 service controls:** Parameters conveyed as part of a directory operation
18 that constrain various aspects of its performance. [ISO/IEC 9594-3: 1990 {B8}]
- 19 **2.2.2.77 session:** A sequence of directory operations requested by a particular
20 user of a particular DUA using the same session OM object. [IEEE Std 1224.2-
21 1993 {4}]
- 22 **2.2.2.78 signed:** The condition of information that has an enciphered summary
23 appended to it that is used to ensure the integrity of the data, the authenticity of
24 the originator, and the unambiguous relationship between the originator and the
25 data. [IEEE Std 1224.2-1993 {4}]
- 26 **2.2.2.79 subordinate:** The converse of superior. [ISO/IEC 9594-2: 1990 {B7}]
- 27 **2.2.2.80 superior** (applying to entry or directory object): Immediately superior,
28 or superior to one that is immediately superior (recursively). [ISO/IEC 9594-2:
29 1990 {B7}]
- 30 **2.2.2.81 syntax:** See *attribute syntax* in 2.2.2.12 and *OM syntax* in 2.2.2.62.
- 31 **2.2.2.82 thread:** A single sequential flow of control within a process. [IEEE Std
32 1224.2-1993 {4}]

2.2.2.83 user: The entity or person that accesses the directory, used in this standard to refer to the application program that is using the interface. [IEEE Std 1224.2-1993 {4}]

2.2.2.84 user certificate [certificate]: The public keys of a user, together with some other information, rendered unforgeable by encipherment with the secret key of the certification authority that issued it. [ISO/IEC 9594-8: 1990 {B12}]

2.2.2.85 value: See *attribute value* in 2.2.2.14 and *OM attribute value* in 2.2.2.59.

2.2.2.86 workspace: A repository for instances of OM classes in the closures of one or more packages associated with the workspace. [IEEE Std 1224.2-1993 {4}]

2.2.3 Abbreviations

For the purposes of this standard, the following abbreviations apply.

2.2.3.1 API: Application Program Interface.

2.2.3.2 ASN.1: Abstract Syntax Notation One.

2.2.3.3 AVA: Attribute Value Assertion.

2.2.3.4 BDCP: Basic Directory Contents Package.

2.2.3.5 BER: Basic Encoding Rules.

2.2.3.6 CA: Certification Authority.

2.2.3.7 CCITT: International Telegraph and Telephone Consultative Committee.

2.2.3.8 DAP: Directory Access Protocol.

2.2.3.9 DIB: Directory Information Base.

2.2.3.10 DIT: Directory Information Tree.

2.2.3.11 DL: Delivery List.

2.2.3.12 DMD: Directory Management Domain.

2.2.3.13 DN: Distinguished Name.

- 1 **2.2.3.14 DS:** Directory Service.
- 2 **2.2.3.15 DS API:** Directory Services API (as defined in this standard).
- 3 **2.2.3.16 DS Package:** Directory Service Package.
- 4 **2.2.3.17 DSA:** Directory System Agent.
- 5 **2.2.3.18 DSP:** Directory System Protocol.
- 6 **2.2.3.19 DUA:** Directory User Agent.
- 7 **2.2.3.20 IA5:** International Alphabet No. 5.
- 8 **2.2.3.21 ID:** Identifier.
- 9 **2.2.3.22 the Interface:** The interface to directory services defined in this
10 standard.
- 11 **2.2.3.23 ISDN:** Integrated Services Digital Network.
- 12 **2.2.3.24 ISO:** International Organisation for Standardization.
- 13 **2.2.3.25 MDUP:** MHS Directory User Package.
- 14 **2.2.3.26 MHS:** Message Handling System (as in ISO/IEC 10021: 1990).
- 15 **2.2.3.27 MS:** Message Store.
- 16 **2.2.3.28 NBP:** Nonbasic Parameter.
- 17 **2.2.3.29 O/R:** Originator/Recipient.
- 18 **2.2.3.30 OM:** OSI abstract data Manipulation.
- 19 **2.2.3.31 OSI:** Open Systems Interconnection.
- 20 **2.2.3.32 ISO/IEC 9945-1:** ISO/IEC 9945-1: 1990 {2}.
- 21 **2.2.3.33 POSIX.3:** IEEE Std 1003.3-1991 {B13}.
- 22 **2.2.3.34 RDN:** Relative Distinguished Name.

- 1 **2.2.3.35 X.500 directory standards:** ISO/IEC 9954: 1990 and the 1988 CCITT
- 2 X.500 Series recommendations ({B6}–{B12}).
- 3 **2.2.3.36 SAP:** Strong Authentication Package.

Section 3: The Generic Interface

3.1 Introduction

NOTE: This section defines the C language binding corresponding to the interface operations, interface class definitions, and error class definitions of the language-independent specification in IEEE Std 1224.2-1993 {4}. The C language binding corresponding to the interface operations is in 3.2. The C language binding corresponding to the interface class definitions and error class definitions is in 3.3. These subclauses are followed by a subclause on the `<xds.h>` header.

Some of the C language datatypes appearing in this section are defined in IEEE Std 1327-1993 {6}. The types in question are those whose names begin with “OM_”. For the purpose of this standard, the definitions given in IEEE Std 1327-1993 {6} shall apply.

In the C binding, instances of classes defined in IEEE Std 1224.2-1993 {4} shall be represented by data structures of C language type *OM_object*, defined in IEEE Std 1327-1993 {6}.

NOTE: No additional type definitions for these classes are contained in this section.

All object identifiers shall be represented by constants in the C binding. These constants can be used with the macros defined in IEEE Std 1327-1993 {6}.

NOTE: Constants representing object identifiers required for the Directory Service package are defined in this section.

Every application program that makes use of a class or other object identifier name shall explicitly import it into every compilation unit (C source program) that uses it. Each such class or object identifier name shall be explicitly exported from just one compilation unit.

NOTE: The descriptions of the OM_IMPORT and OM_EXPORT macros in IEEE Std 1327-1993 {6} contain an explanation.

3.2 Interface Functions

3.2.1 Overview

For each operation of IEEE Std 1224.2-1993 {4}, there shall be a corresponding function of the C binding. Each C function shall have the same name as the operation to which it corresponds.

Except for *ds_initialize()* and *ds_version()*, the arguments of each C function shall correspond to the parameters of the corresponding language-independent specification operation. For input parameters, the function argument name shall be the same as the parameter name. For output parameters, the function argument name shall be derived from the parameter name by appending “_return” to it.

The processing performed by each C language function shall be exactly as described for the corresponding operation of IEEE Std 1224.2-1993 {4}.

Except for *ds_initialize()*, the C function shall return a result of C language type *DS_status*, corresponding to the status datatype *ds_status_type* of the equivalent operation of IEEE Std 1224.2-1993 {4}.

The error indications that each C function can return shall correspond to the errors described for the equivalent operation of IEEE Std 1224.2-1993 {4}.

NOTES:

- (1) Except for *ds_initialize()* and *ds_version()*, this section contains synopses of the C functions but no detailed descriptions of them.
- (2) Descriptions of the language-independent operations are contained in IEEE Std 1224.2-1993 {4}.

3.2.2 Common Datatypes

```
typedef OM_private_object DS_status;
```

The datatype of most of the C functions defined in this section, corresponding to the datatype *ds_status_type* of IEEE Std 1224.2-1993 {4}.

3.2.3 Abandon

```
DS_status ds_abandon(  
    OM_private_object session,  
    OM_sint invoke_id  
);
```